{EPITECH}

JAVA SEMINAR_

< DAY 06 - EVERYTHING TOGETHER />

Unless specified otherwise, all messages must be followed by a newline and the names of the getter and setter for `Attribute` will always be like `getAttribute` and `setAttribute`.
For instance, attribute `Bobby` will have `getBobby` and `setBobby`.
FYI, this name convention is known as *CamelCase*.

Exercise 01

Delivery: ./Character.java

Create an abstract Character class with the following **protected** attributes: name, life, agility, strength, wit, and a constant RPGClass string attribute, with the corresponding getters.

These attributes must have the following values by default:

✓ name: first argument passed to constructor
✓ RPGClass: second argument passed to constructor
✓ life: 50
✓ agility: 2
✓ strength: 2
✓ wit: 2

Add an attack method that:

✓ takes a string as argument ;
✓ prints [Character's name]: Rrrrrrrr.... whatever the argument is.

Here is an example in which TestCharacter is an implementation of your abstract class:

```java
public class Example {
    public static void main(String[] args) {
        Character perso = new TestCharacter("Jean-Luc");

        System.out.println(perso.getName());
        System.out.println(perso.getLife());
        System.out.println(perso.getAgility());
        System.out.println(perso.getStrength());
        System.out.println(perso.getWit());
        System.out.println(perso.getRPGClass());

        perso.attack("my weapon");
    }
}
```

Terminal

```
$> java Example
Jean-Luc
50
2
2
2
SomethingSetByTestCharacter
Jean-Luc: Rrrrrrrr....
```

Exercise 02

Delivery: `./Character.java`, `./Warrior.java`, `./Mage.java`

Create the `Warrior` and `Mage` classes, which **extend** the `Character` class. Modify attributes as follows:

attribute	`Warrior`	`Mage`
`RPGClass`	Warrior	Mage
`life`	100	70
`strength`	10	3
`agility`	8	10
`wit`	3	10

Implement the `attack` method for the 2 classes, its parameter defines the weapon used.
Your characters are proud and like to announce themselves. At creation, display:

- ✓ `[name]: My name will go down in history!` when creating a `Warrior` object;
- ✓ `[name]: May the gods be with me.` when creating a `Mage` object.

A warrior can **only** attack with a `hammer` or a `sword`. The `Warrior` class's `attack` method displays:

```
[name]: Rrrrrrrr....
[name]: I'll crush you with my [weapon]!
```

A mage can **only** attack with `magic` or a `wand`. The `Mage` class's `attack` method displays:

```
[name]: Rrrrrrrr....
[name]: Feel the power of my [weapon]!
```

Here is an example:

```java
public class Example {
    public static void main(String[] args) {
        Character warrior = new Warrior("Jean-Luc");
        Character mage = new Mage("Robert");
        warrior.attack("hammer");
        mage.attack("magic");
    }
}
```

```
Terminal
$> java Example
Jean-Luc: My name will go down in history!
Robert: May the gods be with me.
Jean-Luc: Rrrrrrrr....
Jean-Luc: I'll crush you with my hammer!
Robert: Rrrrrrrr....
Robert: Feel the power of my magic!
```

Exercise 03

Delivery: `./Character.java`, `./Warrior.java`, `./Mage.java`, `./Movable.java`

We now have characters who can be Mages or Warriors.
They can attack, fair enough, but they still cannot move! This is bothersome...

In order to add this behavior to your classes, create an **interface** called `Movable` that contains the following methods: `moveRight`, `moveLeft`, `moveForward` and `moveBack`.

This interface will obviously be implemented by the `Character` classes.

Each method displays one (guess which!) of these messages:

- ✓ `[name]: moves right` ;
- ✓ `[name]: moves left` ;
- ✓ `[name]: moves forward` ;
- ✓ `[name]: moves back`.

Exercise 04

Delivery: `./Character.java`, `./Warrior.java`, `./Mage.java`, `./Movable.java`

Paralysis is over! Your characters can now move, but, being so proud, they want more!
Your boorish Warrior refuses to be compared to a small and skinny Mage.
While the Warrior moves in a bold and virile manner, the Mage moves delicately!

To satisfy your Warrior, implement overrides for the `Movable` methods inherited by `Character`.
Each method displays a message that correspond to the class that overrides them:

Warrior	Mage
`[name]: moves right like a bad boy.`	`[name]: moves right furtively.`
`[name]: moves left like a bad boy.`	`[name]: moves left furtively.`
`[name]: moves back like a bad boy.`	`[name]: moves back furtively.`
`[name]: moves forward like a bad boy.`	`[name]: moves forward furtively.`

Here is an example:

```java
public class Example {
    public static void main(String[] args) {
        Warrior warrior = new Warrior("Jean-Luc");
        Mage mage = new Mage("Robert");

        warrior.moveRight();
        warrior.moveLeft();
        warrior.moveBack();
        warrior.moveForward();

        mage.moveRight();
        mage.moveLeft();
        mage.moveBack();
        mage.moveForward();
    }
}
```

Terminal

```
$> java Example
Jean-Luc: My name will go down in history!
Robert: May the gods be with me.
Jean-Luc: moves right like a bad boy.
Jean-Luc: moves left like a bad boy.
Jean-Luc: moves back like a bad boy.
Jean-Luc: moves forward like a bad boy.
Robert: moves right furtively.
Robert: moves left furtively.
Robert: moves back furtively.
Robert: moves forward furtively.
```

Exercise 05

Delivery: `./Character.java`, `./Warrior.java`, `./Mage.java`, `./Movable.java`

Your characters are now customized to talk, walk and attack. Being able to attack is nice, but attacking while the weapon is still in its sheath is going to be difficult...

You will agree that, whether Warrior or Mage, the character will draw his weapon the same way.

Make sure the `Character` class implements the `unsheathe` method/
Doing so, both `Warrior` and `Mage` will inherit from it.

However, make sure the `unsheathe` method cannot be override by `Warrior` and `Mage`.

Display `[name]: unsheathes his weapon.` when the method is called.

Exercise 06

Delivery: `./exceptions/Character.java`, `./exceptions/Warrior.java`, `./exceptions/Mage.java`, `./exceptions/Movable.java`, `./exceptions/WeaponException.java`

Copy your previous classes in a directory called `exceptions`.

This directory do not act as a `package`. The java classes within it **must** be part of the default package and not some kind of `exceptions` one.

Let's create a `WeaponException` class dedicated to weapons error management, inheriting from the `Exception` class, in which at least two different messages must be declared:

- ✓ `[name]: I refuse to fight with my bare hands.` if `attack` parameter is empty ;
- ✓ `[name]: A [weapon]?? What should I do with this?!` if warrior `attack` parameter is inappropriate ;
- ✓ `[name]: I don't need this stupid [weapon]! Don't misjudge my powers!` if the mage `attack` parameter is inappropriate.

The `attack` method must **throw** a `WeaponException` with the appropriated message in case of errors.

Implement a new method `tryToAttack` in order to:

- ✓ call the `attack` method ;
- ✓ catch the exception ;
- ✓ print the message.

Here is an example:

```java
public class Example {
    public static void main(String[] args) {
        Character warrior = new Warrior("Jean-Luc");
        Character mage    = new Mage("Robert");

        warrior.tryToAttack("screwdriver");
        mage.tryToAttack("hammer");
        warrior.tryToAttack("hammer");
        try {
            mage.attack("");
        }
        catch (WeaponException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Terminal

```
$> java Example
Jean-Luc: My name will go down in history!
Robert: May the gods be with me.
Jean-Luc: A screwdriver?? What should I do with this?!
Robert: I don't need this stupid hammer! Don't misjudge my powers!
Jean-Luc: Rrrrrrrrr....
Jean-Luc: I'll crush you with my hammer!
Robert: I refuse to fight with my bare hands.
```

v 3.4.1

{EPITECH}